Programmation fonctionnelle TP 6

antoine.frenoy@inserm.fr
http://perso.crans.org/frenoy/caml2013.html

Jeudi 28 février 2013

On part du principe que vous maîtrisez le programme des semaines précédentes, et donc que vous êtes capables de refaire les exercices des TD et TP sans problème majeur. Si ce n'est pas le cas, commencez par réviser les bases (types, fonctions, n-uplets, listes, filtrage) avant de passer à cette feuille d'exercices.

Récursivité sur les entiers

Fonctions simples

1. Écrivez une fonction récursive calculant la factorielle d'un entier.
2. Écrivez une fonction récursive calculant le coefficient binomial $\binom{n}{k}$.
3. Écrivez une fonction récursive calculant le `n`ième terme de la suite de Fibonacci.

Fonction d'Ackermann

Soit la fonction $A : \mathbb{N} * \mathbb{N} \to \mathbb{N}$ définie par :
- $A(0, n) = n + 1$
- $A(m, 0) = A(m - 1, 1)$ si $m > 0$
- $A(m, n) = A(m - 1, A(m, n - 1))$ si $m > 0$ et $n > 0$

Écrivez une fonction prenant en arguments deux entiers `m` et `n` et renvoyant $A(\texttt{m}, \texttt{n})$.

Primalité

1. Définissez une fonction `pgcd` qui prend en arguments deux entiers et renvoie leur PGCD.
2. Définissez une fonction `isprime` qui prend en argument un entier positif `n` et indique s'il est premier. On pourra pour cela définir une fonction intermédiaire prenant en arguments deux entiers `n` et `i` et indiquant si il existe un entier compris entre `2` et `i` qui divise `n`.

Suite récursive

Écrivez une fonction prenant en arguments un flottant `a` et une fonction `f: int -> int` ; et renvoyant une fonction prenant en argument un entier `n` et renvoyant le `n`ième terme de la suite définie par :
- $u_0 = \texttt{a}$
- $u_{n+1} = \texttt{f}\ u_n$

Exponentiation rapide

Soit $x \in \mathbf{R}$ et $n \in \mathbf{N}$. On a la relation suivante :
- Si n est pair alors $x^n = (x^{\frac{n}{2}})^2$
- Sinon $x^n = x * (x^{\frac{n-1}{2}})^2$

1. Écrivez une fonction récursive `exporapide` prenant deux arguments `x` et `n` et renvoyant `x` exposant `n` en utilisant la méthode décrite ci-dessus.
2. Quel est son avantage par rapport à la méthode "naïve" vue la semaine précédente ?

Récursivité sur les listes

Fonctions simples

1. Écrivez une fonction récursive prenant un argument `x` de type `'a` et un argument `l` de type `'a list` ; et indiquant si `x` appartient à `l`. On n'utilisera évidemment pas `List.mem`.
2. Écrivez une fonction récursive prenant en argument une liste et renvoyant la liste "miroir".

Sous listes

1. Définissez une fonction `eqlist` qui prend en arguments deux listes et indique si elles sont identiques.
2. Définissez une fonction `sublist` qui prend en arguments deux listes `a` et `b` et indique si `a` est une sous-liste de `b`.

Tri

1. Définissez une fonction `first` qui prend en argument un couple et renvoie son premier élément, et une fonction `second` qui prend en argument un couple et renvoie son deuxième élément.
2. Définissez une fonction `minn` qui prend en argument une liste et renvoie son plus petit élément ainsi que la position de ce plus petit élément dans la liste (la tête de la liste étant en position `0`).
3. Définissez une fonction `suppr` qui prend en arguments une liste `l` et un entier `i` et renvoie une liste identique à la liste `l` sans son `ième` élément.
4. Définissez une fonction `tri` qui prend en argument une liste d'entiers `l` et la trie par ordre croissant. On n'utilisera évidemment pas `List.sort`.

Sommes de listes

1. Écrivez une fonction récursive prenant en arguments deux listes d'entiers et calculant leur somme alignée sur le terme de droite.
2. Écrivez une fonction récursive prenant en arguments deux listes d'entiers et calculant leur somme alignée sur le terme de gauche.

Fonctions moins simples

1. Écrivez une fonction concaténant deux listes, bien sûr sans utiliser `List.append`.
2. Écrivez une fonction prenant en arguments un prédicat `f` de type `'a -> bool` et une liste `l` de type `'a list` et renvoyant la liste des élements de `l` qui verifient `f`.

Distances

1. La distance de Hamming indique, pour deux listes de caractères de même longueur, le nombre de positions où les deux listes diffèrent, c'est à dire le nombre de `i` tels que le `ième` caractère de la première liste soit différent du `ième` caractère de la deuxième liste. Écrivez une fonction `hamming` prenant en arguments deux listes de caractères et renvoyant leur distance de Hamming.
2. La distance de Levenshtein indique, pour deux listes de caractères (de longueurs éventuellement différentes) le nombre minimal de caractères qu'il faut supprimer, insérer ou remplacer pour passez d'une liste à l'autre. Écrivez une fonction `levenshtein` prenant en arguments deux listes de caractères et renvoyant leur distance de Levenshtein. On pourra définir une fonction intermédiaire `min3` qui renvoie le minimum de trois éléments.